**Gerenciamento de memória implementação de simulador de MMU**
**Proj2**

**Informações**

- Trabalho em duplas.
- Entrega no Portal COMP<`http://trab.dc.unifil.br/moodle/`>.

# 1 Introdução

Neste projeto, vamos implementar um terminal REPL (Read-eval-print loop)[1] que interpretará comandos de criação e término de processos, e eventos de acessos à memória, para simularmos o funcionamento de MMUs (Memory Management Units) e algoritmos de sistemas operacionais para gerenciamento de memória.

# 2 Ferramentas e padronização de entrega

Este trabalho poderá ser feito na linguagem de programação de sua preferência, conquanto que o programa possa ser rodado em qualquer dos principais sistemas operacionais, Windows, Linux e macOS. A única obrigação dos alunos é escrever um guia de compilação e execução do programa na forma de `README.md` (arquivo texto no formato *markdown*), e utilizar um *build system* que possa ser executado com até 3 ou 4 comandos em terminal. Exemplos: `make`, `maven`, `ant`, `rake`, `gradle`, `grunt`, `nuget`, etc.

Na dúvida, utilize Java com IntelliJ, criando um projeto Maven do banco de modelos da IDE. Ainda assim, antes de entregar, se certifique que é possível compilar e rodar a partir do terminal, utilizando comandos `maven`.

Entregar o projeto contendo os códigos-fonte, o arquivo de intruções `README.md` e os arquivos de projeto do *build system* (ex. `pom.xml` para maven), compactados como `zip`. Não inclua binários compilados, nem arquivos intermediários, como logs. Falha em cumprir essas exigência resultará em desconto de pontos.

# 3 Roteiro de atividades

Antes de iniciar qualquer implementação, leia todo o roteiro de atividades, para você ter visão abrangente dos requisitos, e aumentar as chances de projetar uma boa arquitetura de software.

A seguir, as atividades que devem ser desenvolvidas neste trabalho:

1. Implemente o prompt do terminal REPL, utilizando o caractere `>` seguido de um espaço em branco para indicar que o sistema está pronto para receber um novo comando do usuário. Programe também o comando `ajuda`, que exibe uma lista de todos os comandos que o sistema entende, e ainda indique ao usuário quais desses comandos estão de fato implementados.

[1] Quem desejar desafio extra, pode fazer uma GUI.

2. Agora, implemente o comando `configurar`, que cria um novo ambiente de simulação de gerenciamento de memória. O comando recebe os seguintes parâmetros, necessariamente na ordem

   i. quantidade de bits do espaço de memória;

   ii. quantidade de memória física instalada, em palavras, e reconhecendo entradas em base decimal e hexadecimal (exemplo, `65536` e `0x10000` denotam a mesma quantidade de palavras);

   iii. Tipo de MMU do sistema, que pode ser

   - registradores base e limite, com algoritmo de *swapping* por
     - bitmap, seguido pelo tamanho de cada pedaço de alocação de memória;
     - ou, lista encadeada.
   - ou, memória virtual
     (a) quantidade de bits a partir do MSB (most significant bit) reservados para indexar páginas de um processo.

   O programa deverá validar se os parâmetros do comando fazem sentido entre si. Por exemplo, o terminal deverá retornar erro, com indicação do ocorrido, se o tipo de MMU for "memória virtual" com lista encadeada. Outro exemplo é caso o usuário indique mais memória física instalada do que a capacidade de endereçamento de memória, ou uma quantidade de memória instalada menor que o tamanho de um único quadro (o que inviabiliza o sistema).

3. Implemente o comando `processo`, que cria um novo processo no nosso sistema simulado. Como parâmetros, é necessário indicar um número *pid* do novo processo.

   Caso a simulação atual seja de um sistema com swapping

   (a) é necessário indicar tamanho de memória do novo processo no comando `processo`. Se o tamanho do processo for maior que a memória instalar, recusar o comando com um aviso para o usuário. Em seguida,

   (b) utilizar o algoritmo de *first-fit* para encaixar o processo na memória. Porém, se o processo não couber em lugar algum

      i. fazer relocação de todos os processos atuais na memória para gerar um único espaço contínuo de memória livre; mas

      ii. se ainda assim não couber o novo processo, realizar o swapping, a escrita de um processo, o mais antigo, para o HD e sua retirada da memória. Repita esses dois passos o quanto for necessário para que o processo caiba na memória.

4. Implemente o comando `acesso`, que simula o acesso à memória por um processo, e tem como parâmetros obrigatórios o *pid* e a posição da memória (em decimal ou hexadecimal). O comando ainda deve considerar os seguintes fatores:

   (a) Caso esteja-se simulando o swapping, verificar se o acesso foi ilegal, e em caso afirmativo, o processo é automaticamente encerrado, e o terminal avisa o ocorrido ao usuário.

(b) Ainda em caso de swapping, se o processo estiver no disco, o sistema recusa o comando e gera uma mensagem de erro para o usuário.

(c) Em caso de memória virtual, se o acesso causar uma falha de página, o sistema deverá informar os detalhes para o usuário: página acessada que causou falha, quadro liberado e onde consequentemente a página foi posicionada.

5. Implemente o comando `terminar`, que termina espontaneamente o processo identificado por um *pid*, parâmetro obrigatório do comando. A memória, os registros e a tabela de processos deve ser totalmente limpa do processo terminado.

6. Implemente o comando `relatorio`, que faz com que o terminal exiba todo o estado de simulação, que inclui processos existentes e alocação de memória.